**Лекция 10 Файлы и разрешения**

В этой лекции обсуждается работа [с файлами](https://developer.android.com/training/basics/data-storage/files.html) в Android. Использование файловой системы позволяет нам иметь [постоянное хранилище данных](https://developer.android.com/guide/topics/data/data-storage.html) более обширным и гибким способом, чем использование SharedPreferencesобсуждавшегося в предыдущей лекции (и в качестве дополнения к ContentProviderбазам данных).

В этой лекции используется код, найденный по адресу <https://github.com/info448/lecture10-files> . Для демонстрации всех функций, обсуждаемых в этой лекции, ваше устройство или эмулятор должны работать под управлением **API 23 (6.0 Marshmallow)** или более поздней версии.

**10.1 Места хранения файлов**

Устройства Android разделяют хранилище файлов на два типа: **внутреннее хранилище** и **внешнее хранилище** . Эти названия появились, когда устройства имели встроенную память, а также внешние SD-карты, каждая из которых могла иметь разные взаимодействия. Однако в современных системах «внешнее хранилище» может также относиться к разделу встроенной памяти телефона; различия вместо этого используются для указания *доступа,* а не физического расположения данных.

* [**Внутреннее хранилище**](https://developer.android.com/guide/topics/data/data-storage.html#filesInternal) всегда доступно, и по умолчанию файлы, сохраненные внутри,*доступны только* вашему приложению. Аналогично, когда пользователь удаляет ваше приложение, внутренние файлы удаляются. Обычно это лучшее место для «личных» файловых данных или файлов, которые будут использоваться только вашим приложением.
* [**Внешнее хранилище**](https://developer.android.com/guide/topics/data/data-storage.html#filesExternal) не всегда доступно (например, если физическое хранилище удалено) и обычно (но не всегда)*доступно для чтения всем* . Обычно файлы, хранящиеся во внешнем хранилище, сохраняются даже при удалении приложения, если не используются определенные параметры. Обычно это используется для «публичных» файлов, которые могут совместно использоваться приложениями.

[Когда мы используем каждый из них?](https://developer.android.com/training/basics/data-storage/files.html#InternalVsExternalStorage)[34](https://info448.github.io/files-and-permissions.html#fn34) По сути, вам следует использовать *внутреннее* хранилище для «личных» файлов, которые вы не хотите, чтобы были доступны за пределами приложения, а в противном случае используйте *внешнее* хранилище.

* Однако следует отметить, что существуют публично **скрытые***внешние* файлы — главное различие между местами хранения заключается в меньшей видимости и большей сложности *с доступом* .

Кроме того, обе эти системы хранения также имеют местоположение **«кэша»** (т. е. *внутренний кэш* и *внешний кэш* ). [Кэш](https://en.wiktionary.org/wiki/cache) — это «(секретное) хранилище для будущего», но в вычислительной технике его обычно называют «временным хранилищем». Кэши отличаются от других хранилищ файлов тем, что Android имеет возможность автоматически удалять кэшированные файлы, если место на диске становится мало. Однако вы не можете полагаться на то, что операционная система сделает это сама эффективным способом, поэтому вам все равно следует удалять собственные файлы кэша, когда вы закончите с ними работать! Короче говоря, используйте кэши для временных файлов и старайтесь, чтобы они были *небольшими* (рекомендуется менее 1 МБ).

* Пользователь также может легко очистить кэш приложения через пользовательский интерфейс операционной системы.

В коде использование всех этих мест хранения подразумевает работу с классом [35.](https://info448.github.io/files-and-permissions.html#fn35) Этот класс представляет объект «файл» (или «каталог») и является тем же классом, с которым вы, возможно, знакомы по Java SE.[File](https://developer.android.com/reference/java/io/File.html)

* Мы можем создать экземпляр a, Fileпередав ему каталог (который является другим File) и имя файла (a String). Создание экземпляра файла создаст файл на диске (но пустой, размером 0), если он еще не существует.
* Мы можем проверить, Fileявляется ли a папкой с помощью .isDirectory()метода, и создать новые каталоги, взяв a Fileи вызвав .mkdir()его. Мы можем получить список Filesвнутри каталога с помощью listFiles()метода. Более подробную информацию и параметры см. в документации API.

Разница между сохранением файлов во Внутреннем и Внешнем хранилище ***на практике*** заключается просто в том, в какой каталог вы помещаете файл! В этой лекции основное внимание будет уделено работе с **Внешним хранилищем** , поскольку этот код в конечном итоге становится своего рода «супер-набором» деталей реализации, необходимых для файловой системы в целом. Он укажет, какие изменения необходимо внести для взаимодействия с Внутренним хранилищем.

* В частности, в этой лекции будет рассмотрен процесс реализации приложения, которое будет сохранять в файл все, что пользователь вводит в текстовое поле.

Поскольку внешнее хранилище устройства может находиться на съемном носителе, для того, чтобы взаимодействовать с ним каким-либо образом, нам сначала нужно проверить, доступно ли оно (например, что SD-карта смонтирована). Это можно сделать с помощью следующей [проверки](https://developer.android.com/guide/topics/data/data-storage.html#MediaAvail) (написанной как вспомогательный метод, чтобы ее можно было использовать повторно):

*//java*

**public** static boolean isExternalStorageWritable() {

String state = Environment.getExternalStorageState();

**if** (Environment.MEDIA\_MOUNTED.equals(state)) {

**return** **true**;

}

**return** **false**;

}

*//kotlin*

**fun** isExternalStorageWritable(): Boolean {

**return** Environment.getExternalStorageState() == Environment.MEDIA\_MOUNTED

}

**10.2 Разрешения**

Прямой доступ к файловой системе любого компьютера может представлять существенную угрозу безопасности, поэтому существуют существенные меры защиты, гарантирующие, что вредоносное приложение не будет бесчинствовать над данными пользователя. Чтобы работать с файловой системой, нам сначала нужно более подробно обсудить, как Android обрабатывает [разрешения .](https://developer.android.com/guide/topics/permissions/requesting.html)

Одним из наиболее важных аспектов дизайна операционной системы Android является идея [**песочницы**](https://en.wikipedia.org/wiki/Sandbox_(computer_security)) : каждое приложение получает свою собственную «песочницу» для игры (где хранятся все его игрушки), но не может выйти за пределы коробки и поиграть с чужими игрушками. В Android «игрушки» (компоненты), которые находятся за пределами песочницы, — это то, что может *повлиять* на пользователя, например, доступ к сети или файлам. Приложения не на 100% заперты в своей песочнице, но нам нужно проделать дополнительную работу, чтобы выйти за ее пределы.

* «Песочница» также происходит на уровне пакетов, где пакеты (приложения) изолируются от пакетов *других разработчиков* ; вы можете использовать подписание сертификата (которое происходит автоматически в процессе сборки), чтобы пометить два пакета как от одного разработчика, если вы хотите, чтобы они взаимодействовали.
* Кроме того, базовая операционная система Android основана на Linux, поэтому она фактически использует под капотом систему разрешений Linux (с идентификаторами пользователей и групп, которые предоставляют доступ к определенным файлам или процессам).

Чтобы приложение могло выйти за пределы своей песочницы (и использовать другие компоненты), ему необходимо запросить разрешение на выход. Мы запрашиваем это разрешение («Мама, можно?»), явно объявляя использование вне песочницы в Manifest, как мы делали раньше, получая разрешение на доступ в Интернет, отправку SMS-сообщений или доступ к пользовательскому словарю.

Однако разрешения Android, которые мы можем запросить, делятся на [две категории](https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous) : **обычные** и **опасные** :

* **Обычные разрешения** — это те, которые могут повлиять на пользователя (поэтому требуют разрешения), но не представляют серьезного риска. Они предоставляются пользователем во *время установки* ; если пользователь решает установить приложение, разрешение предоставляется этому приложению. См. [этот список](https://developer.android.com/guide/topics/permissions/normal-permissions.html)[36](https://info448.github.io/files-and-permissions.html#fn36) для примеров обычных разрешений. INTERNET— это обычное разрешение.
* **Опасные разрешения** , с другой стороны, несут риск нарушения конфиденциальности пользователя или иного вмешательства в устройство пользователя или другие приложения. Эти разрешения *также* должны быть предоставлены во время установки. НО ***В ДОПОЛНЕНИЕ*** , начиная с API 23 (Android 6.0 Marshmallow), пользователи *дополнительно* должны предоставлять опасные разрешения **во время выполнения** , когда приложение пытается фактически вызвать «разрешенное» опасное действие.
  + Пользователь предоставляет разрешение через всплывающее диалоговое окно, сгенерированное системой. Обратите внимание, что разрешения предоставляются в «группах», поэтому, если пользователь соглашается дать вам RECEIVE\_SMSразрешение, вы SEND\_SMSтакже получаете разрешение. См. [список групп разрешений](https://developer.android.com/guide/topics/permissions/requesting.html#perm-groups) .
  + Когда пользователь предоставляет разрешение во время выполнения, это разрешение остается предоставленным, пока приложение установлено. Но большая оговорка в том, что пользователь может в ***любой* момент отозвать** или отклонить привилегии (они делают это через системные настройки)! Таким образом, вам придется проверять *каждый раз, когда вы хотите получить доступ к функции* , предоставил ли пользователь привилегии или нет — вы не знаете, предоставил ли пользователь вам разрешение *в данный момент* , даже если он это сделал в прошлом.

Запись на внешнее хранилище — *опасное* разрешение, поэтому нам придется проделать дополнительную работу для поддержки системы разрешений среды выполнения Marshmallow.

* Для поддержки разрешений времени выполнения нам нужно указать **целевой SDK** нашего приложения как 23или выше И запустить приложение на устройстве под управлением Android 6.0 (Marshmallow) или выше. Разрешения времени выполнения учитываются только в том случае, если ОС поддерживает *и* приложение нацелено на это. Для устройств или приложений с более низким API разрешение предоставляется только во время установки.

Сначала нам *все еще* нужно запросить разрешение в Manifest; если мы не объявили, что можем запросить разрешение, нам не будет разрешено запрашивать его в будущем. В частности, сохранение файлов во внешнем хранилище требует android.permission.WRITE\_EXTERNAL\_STORAGEразрешения (которое также предоставит нам READ\_EXTERNAL\_STORAGEдоступ).

Прежде чем совершить опасное действие, мы можем проверить, есть ли у нас на данный момент разрешение:

*//java*

int permissionCheck = ContextCompat.checkSelfPermission(activity, Manifest.permission.PERMISSION\_NAME);

*//kotlin*

**val** permissionCheck = ContextCompat.checkSelfPermission(**this**, Manifest.permission.PERMISSION\_NAME)

* Эта функция в основном «смотрит», предоставлено ли приложению *в настоящее время* определенное разрешение или нет. Она вернет либо , PackageManager.PERMISSION\_GRANTEDлибо PackageManager.PERMISSION\_DENIED.

Если разрешение было предоставлено, отлично! Мы можем заниматься своими делами (например, сохранять файл на внешнем хранилище). Но если разрешение НЕ было предоставлено явно (во время выполнения), то нам придется его запросить. Мы делаем это, вызывая:

*//java*

ActivityCompat.requestPermissions(activity, **new** String[]{Manifest.permission.PERMISSION\_NAME}, REQUEST\_CODE);

*//kotlin*

ActivityCompat.requestPermissions(activity, arrayOf(Manifest.permission.PERMISSION\_NAME), REQUEST\_CODE)

* Этот метод принимает Context и *массив* разрешений, к которым нам нужен доступ (в случае, если нам нужно больше одного). Мы также предоставляем код идентификатора запроса (an int), который мы можем использовать для идентификации этого конкретного запроса на разрешение в обратном вызове, который будет выполнен, когда пользователь выберет, предоставить нам доступ или нет. Это тот же шаблон, что и при отправке Intent для *результата* ; запрос разрешения концептуально похож на отправку Intent в систему разрешений!

Затем мы можем предоставить обратный вызов, который будет выполнен, когда пользователь решит, предоставить нам разрешение или нет:

*//java*

**public** void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {

**switch** (requestCode) {

**case** REQUEST\_CODE:

**if** (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

*//have permission! Do stuff!*

}

**default**:

**super**.onRequestPermissionsResult(requestCode, permissions, grantResults);

}

}

*//kotlin*

**override** **fun** onRequestPermissionsResult(requestCode: Int, permissions: Array<String>, grantResults: IntArray) {

**when** (requestCode) {

REQUEST\_CODE -> {

**if** (grantResults.size > 0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

*//have permission! Do stuff!*

}

}

**else** -> **super**.onRequestPermissionsResult(requestCode, permissions, grantResults)

}

}

Мы проверяем, по какому запросу мы слышим результаты, какие разрешения были предоставлены (если таковые имеются — пользователь может предоставлять разрешения по частям!), а затем можем отреагировать, если все хорошо... например, наконец-то сохранить наш файл!

* Обратите внимание, что если пользователь отказывает нам в разрешении один раз, мы можем попытаться объяснить, *почему* мы просим разрешения (см. [лучшие практики](https://developer.android.com/training/permissions/best-practices.html) ) и попросить снова. Google предлагает метод утилиты ( ActivityCompat#shouldShowRequestPermissionRationale()), который мы можем использовать, чтобы показать диалоговое окно с обоснованием, если они отказали нам один раз. И если это правда, мы можем показать Диалог или что-то еще, чтобы объяснить себя — и если они одобрят этот диалог, то мы можем попросить выполнить опасное действие снова.

**10.3 Внешнее хранилище**

Как только у нас появится разрешение на запись во внешний файл, мы действительно сможем это сделать! Поскольку мы убедились, что внешнее хранилище доступно, теперь нам нужно выбрать, в какой каталог в этом хранилище сохранить файл. С внешним хранилищем у нас есть два варианта:

* Мы можем сохранить файл [**публично**](https://developer.android.com/guide/topics/data/data-storage.html#SavingSharedFiles) . Мы используем getExternalStoragePublicDirectory()метод для доступа к публичному каталогу, передавая ему нужный нам [тип](http://developer.android.com/reference/android/os/Environment.html#lfields) каталога (например, DIRECTORY\_MUSIC, DIRECTORY\_PICTURES, DIRECTORY\_DOWNLOADSи т. д.). По сути, это помещает файлы в те же папки, которые использует каждое другое приложение, и отлично подходит для общих данных и распространенных форматов, таких как изображения, музыка и т. д. Файлы в публичных каталогах могут быть легко доступны другим приложениям, если у приложения есть разрешение на чтение/запись из внешнего хранилища!
  + DIRECTORY\_DOCUMENTSбыл добавлен в API 19 и является хорошим местом для использования.
* В качестве альтернативы, начиная с API 18, мы можем сохранить файл [**в частном порядке**](https://developer.android.com/guide/topics/data/data-storage.html#AccessingExtFiles) , но все еще на внешнем хранилище (эти файлы *доступны* для чтения всем, но скрыты от пользователя как носители, поэтому они не «выглядят» как публичные файлы). Мы получаем доступ к этому каталогу с помощью getExternalFilesDir()метода, снова передавая ему *тип* (так как мы по сути создаем собственную версию публичных папок). Мы также можем использовать nullдля типа, что даст нам корневой каталог.

Начиная с API 19 (4.4 KitKat), вам не нужно разрешение на запись в *частное* внешнее хранилище. Если это все, что вы делаете, вы можете только указать в манифесте, что вам нужен доступ к внешнему хранилищу для версий ниже этой:

**<uses-permission** android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" android:maxSdkVersion="18" **/>**

Фактически мы можем посмотреть на файловую систему эмулятора и увидеть, что наши файлы создаются с помощью adb. Подключитесь к эмулятору из терминала с помощью adb -s emulator-5554 shell(примечание: adbдолжно быть в вашем PATH). **Публичные** внешние файлы обычно можно найти в /storage/emulated/0/Folder(или /storage/sdcard/Folder), в то время как **частные** внешние файлы можно найти в /storage/emulated/0/Android/data/package.name/files(эти пути могут различаться на разных устройствах).

Открыв файл, мы можем записать в него содержимое, используя те же классы ввода-вывода, которые мы использовали в Java:

* «Низкоуровневый» способ сделать это — создать FileOutputStreamобъект (или FileInputStreamдля чтения). Мы просто передаем этот конструктор для Fileзаписи. Мы пишем bytesв этот поток… но можем получить байты из строки, вызвав myString.getBytes(). Для чтения нам нужно будет считать *все* строки/символы и, возможно, построить из них строку для отображения.
* Однако мы также можем использовать те же *декораторы* , что и в Java (например, BufferedReader, PrintWriterи т. д.), если нам нужны эти возможности; это немного упрощает чтение и запись в файл.
* В любом случае **не забудьте сделать .close()трансляцию, когда закончите** (чтобы избежать утечек памяти)!

*//java*

*//writing*

**try** {

*//saving in public Documents directory*

File dir = getExternalStoragePublicDirectory(Environment.DIRECTORY\_DOCUMENTS);

**if** (!dir.exists()) { dir.mkdirs(); } *//make dir if doesn't otherwise exist (pre-19)*

File file = **new** File(dir, FILE\_NAME);

Log.v(TAG, "Saving to " + file.getAbsolutePath());

PrintWriter out = **new** PrintWriter(**new** FileWriter(file, **true**));

out.println(textEntry.getText().toString());

out.close();

} **catch** (IOException ioe) {

Log.d(TAG, Log.getStackTraceString(ioe));

}

*//reading*

**try** {

File dir = getExternalStoragePublicDirectory(Environment.DIRECTORY\_DOCUMENTS);

File file = **new** File(dir, FILE\_NAME);

**if**(!file.exists()) **return**; *//e.g., if file doesn't exist yet*

BufferedReader reader = **new** BufferedReader(**new** FileReader(file));

StringBuilder text = **new** StringBuilder();

*//read the file*

String line = reader.readLine();

**while** (line != **null**) {

text.append(line + "\n");

line = reader.readLine();

}

textDisplay.setText(text.toString());

reader.close();

} **catch** (IOException ioe) {

Log.d(TAG, Log.getStackTraceString(ioe));

}

*//kotlin*

*//writing*

**try** {

**val** dir = getExternalStoragePublicDirectory(Environment.DIRECTORY\_DOCUMENTS)

**if** (!dir.exists()) { dir.mkdirs() } *//make dir if doesn't otherwise exist (pre-19)*

**val** file = File(dir, FILE\_NAME)

Log.v(TAG, "Saving to " + file.absolutePath)

**val** out = PrintWriter(FileWriter(file, **true**))

**out**.println(textEntry.text.toString())

**out**.close()

} **catch** (ioe: IOException) {

Log.d(TAG, Log.getStackTraceString(ioe))

}

*//reading*

**try** {

**val** dir = getExternalStoragePublicDirectory(Environment.DIRECTORY\_DOCUMENTS)

**val** file = File(dir, FILE\_NAME)

**if** (!file.exists()) **return** *//e.g., if file doesn't exist yet*

**val** reader = BufferedReader(FileReader(file))

**val** text = StringBuilder()

*//read the file*

**var** line: **String**? = reader.readLine()

**while** (line != **null**) {

text.append(line + "\n")

line = reader.readLine()

}

textDisplay.text = text.toString()

reader.close()

} **catch** (ioe: IOException) {

Log.d(TAG, Log.getStackTraceString(ioe))

}

Это позволит нам использовать кнопку «сохранить» для записи сообщения в файл, а кнопку «прочитать» — для загрузки сообщения из файла (и отображения его на экране)!

**10.4 Внутреннее хранилище и кэш**

[Внутреннее хранилище](https://developer.android.com/guide/topics/data/data-storage.html#filesInternal) работает примерно так же, как и внешнее хранилище. Помните, что внутреннее хранилище всегда является *приватным* для приложения. Поэтому нам также не нужно разрешение на доступ к внутреннему хранилищу!

Для Внутреннего хранилища мы можем использовать getFilesDir()метод для доступа к каталогу файлов (так же, как мы делали с Внешним хранилищем). Этот метод обычно возвращает папку в /data/data/package.name/files.

*В качестве альтернативы* мы можем использовать Context#openFileOutput()(или Context#openFileInput()) и передать ему *имя* файла для открытия. Это возвращает нам Streamобъект для этого файла в каталоге файлов внутреннего хранилища, без необходимости выполнять дополнительную работу (исключая посредника!)

* Эти методы принимают второй параметр: MODE\_PRIVATEсоздаст файл (или *заменит* файл с тем же именем). Другие доступные режимы: MODE\_APPEND(который добавляет в конец файла, если он существует, вместо стирания). ~~MODE\_WORLD\_READABLE~~, и ~~MODE\_WORLD\_WRITEABLE~~устарели.
* Обратите внимание, что вы можете обернуть a FileInputStreamв a InputStreamReaderв a BufferedReader.

Мы можем получить доступ к каталогу Internal Cache с помощью getCacheDir(), или к каталогу External Cache с помощью getExternalCacheDir(). Мы почти всегда используем Internal Cache, потому что зачем вам временные файлы, которые могут читать все (кроме, может быть, временных изображений…)

* Рекомендуется создавать временные файлы кэша с помощью [createTempFile()](https://developer.android.com/reference/java/io/File.html#createTempFile(java.lang.String,%20java.lang.String))служебного метода.

И снова, получив файл, вы используете тот же процесс чтения и записи, что и для внешнего хранилища.

**Для практики** сделайте так, чтобы предоставленный переключатель поддерживал чтение и запись во Внутренний файл. Конечно, это будет *другой* файл, нежели тот, который используется с Внешним переключателем. В идеале этот код можно было бы рефакторить, чтобы избежать дублирования, но это становится сложным из-за необходимости в проверенной обработке исключений.

Подводить итоги:

* *Частное внутреннее хранилище* : getFilesDir()илиopenFileOutput()
* *Внешнее публичное хранилище* :getExternalStoragePublicDirectory()
* *Частное внешнее хранилище* :getExternalFilesDir()
* *Внутренний кэш* : getCacheDir()иcreateTempFile()
* *Внешний кэш* :getExternalCacheDir()

**10.5 Пример: сохранение изображений**

В качестве другого примера того, как мы могли бы использовать систему хранения, рассмотрим систему «сделать селфи» из [Лекции 7.](https://info448.github.io/intents.html#intents) Код для съемки фотографии можно найти в отдельном файле PhotoActivity(к которому можно перейти из меню параметров).

Для обзора: мы отправили Intentс MediaStore.ACTION\_IMAGE\_CAPTUREдействием, и *результат* этого Intentвключал *Extra* , который был BitMapминиатюрой низкого качества для изображения. Но если мы хотим сохранить версию этого изображения в более высоком разрешении, нам нужно будет сохранить это изображение в файловой системе!

Для этого мы фактически собираемся изменить то, что Intentмы *отправляем,* так, чтобы оно включало дополнительный Extra: файл, в котором могут быть сохранены данные изображения. По сути, мы заставим *нашу Activity* выделить некоторую память для изображения, а затем сообщим Camera, куда она может поместить данные изображения, которые она запечатлела. (Конверты намерений слишком малы, чтобы переносить целые фотографии!)

Прежде чем отправить Intent, мы создадим (пустой) файл:

*//java*

File file = **null**;

**try** {

String timestamp = **new** SimpleDateFormat("yyyyMMdd\_HHmmss").format(**new** Date()); *//include timestamp*

*//ideally should check for permission here, skipping for time*

File dir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY\_PICTURES);

file = **new** File(dir, "PIC\_"+timestamp+".jpg");

boolean created = file.createNewFile(); *//actually make the file!*

Log.v(TAG, "File created: "+created);

} **catch** (IOException ioe) {

Log.d(TAG, Log.getStackTraceString(ioe));

}

*//kotlin*

**var** file: File? = **try** {

**val** timestamp = SimpleDateFormat("yyyyMMdd\_HHmmss").format(Date()) *//include timestamp*

**val** dir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY\_PICTURES)

**val** file = File(dir, "PIC\_$timestamp.jpg")

file.createNewFile() *//actually make the file!*

file *//return*

} **catch** (ioe: IOException) {

Log.d(TAG, Log.getStackTraceString(ioe))

**null** *//return*

}

* Поскольку мы создаем файл во внешнем хранилище, обязательно сначала спросите разрешение!!

**10.5.1 Файловые провайдеры**

Затем мы укажем дополнительный Extra, чтобы указать камере местоположение этого файла: если мы используем в качестве *ключа*MediaStore.EXTRA\_OUTPUT нашего Extra , камера будет знать, что с этим делать! Однако на самом деле extra будет не Uri **(** напомним: «url» или местоположение файла). Мы отправляем не сам файл, а местоположение ***этого*** файла (потому что это меньшие данные, чтобы поместиться в конверт Intent).File

Вот в чем сложность: если мы попытаемся поделиться абсолютным путем к файлу как Uri, мы получим ошибку, потому что мы «утекаем» информацию — то есть, мы пытаемся позволить другому приложению получить доступ к файлу, на чтение которого у него может не быть разрешения! Нам нужно убедиться, что у Camera действительно есть доступ к месту, где находится файл (он имеет, потому что файл находится в публичном хранилище, но Android этого не знает — он просто видит, что вы делитесь абсолютным путем).

Вместо того, чтобы помещать file://Uri в дополнение к Intent, нам нужно будет создать content://Uri для *ContentProvider* , который может предоставлять файлы любому, кто их запрашивает, независимо от разрешений (поставщик предоставляет разрешение на доступ к своему контенту). A ContentProviderявно означает, что контент доступен вне пакета. В частности, специализированный объект, ContentProviderназываемый a, [FileProvider](https://developer.android.com/reference/android/support/v4/content/FileProvider)может преобразовать набор Filesв набор содержимого данных (например, доступный по content://протоколу), который может использоваться, возвращаться и пониматься другими приложениями!

* В этом отношении это своего рода «файловый сервер»!

Настройка, FileProviderк счастью, не слишком сложна, хотя и состоит из пары шагов. Вам нужно будет объявить внутри <provider>Manifest (см. [ссылку на руководство](https://developer.android.com/training/secure-file-sharing/setup-sharing.html) для примера).

**<provider**

android:name="android.support.v4.content.FileProvider"

android:authorities="edu.uw.myapp.fileprovider"

android:exported="false"

android:grantUriPermissions="true"**>**

**<meta-data**

android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/filepaths" **/>**

**</provider>**

Атрибуты, которые вам необходимо указать:

* android:authorityдолжно быть имя вашего пакета, за которым следует .fileprovider(например, edu.uw.myapp.fileprovider). Это говорит о том, какой источник/домен предоставляет разрешение другим использовать файл.
* Дочерний <meta-data>тег включает android:resourceатрибут, который должен указывать на ресурс XML, типа xml(того же, что используется для SharedPreferences). *Вам нужно будет создать этот файл!* Содержимое этого файла будет списком *подкаталогов* , которые вы хотите, FileProviderчтобы они могли предоставлять. Это будет выглядеть примерно так:

**<?xml** version="1.0" encoding="utf-8"**?>**

**<paths** xmlns:android="http://schemas.android.com/apk/res/android"**>**

*<!--shared from internal storage -->*

**<files-path** path="images/" name="myfiles" **/>**

*<!--shared external storage -->*

**<external-path** path="Documents/" name="documents" **/>**

**<external-path** path="Pictures/" name="pictures" **/>**

**</paths>**

Запись <files-path>ссылается на подкаталог для файлов внутреннего хранилища (то же место, на которое .getFilesDir()указывает ), и <external-path>ссылается на подкаталог для файлов внешнего хранилища. pathАтрибуты указывают имя подкаталога внутри этого места хранения и nameимя, которое FileProvider укажет в качестве пути (представьте: общедоступный путь).

Указав поставщика, вы можете использовать его для доступа Uriк «общей» версии файла с помощью:

Uri fileUri = FileProvider.getUriForFile(context, "edu.uw.myapp.fileprovider", fileToShare);

(обратите внимание, что второй параметр — это «полномочия», которые вы указали в <provider>манифесте). Затем вы можете использовать его Uriкак MediaStore.EXTRA\_OUTPUTдополнительный параметр, чтобы сообщить камере, где сохранять сделанные ею снимки!

* Вы также можете использовать его в качестве EXTRA\_STREAMдополнения в Intent для более общего обмена файлами.
* Затем, когда мы получаем результат изображения от камеры (в нашем onActivityResultобратном вызове), мы можем получить доступ к этому файлу по сохраненному Uri и использовать его для отображения изображения! Это ImageView.setImageUri()быстрый способ отображения файла изображения.

Вы также можете использовать намерение, чтобы убедиться, что фотография [добавлена ​​в Галерею телефона](https://developer.android.com/training/camera/photobasics#TaskGallery) .

Обратите внимание, что при работе с изображениями у нас может очень быстро закончиться память (потому что изображения могут быть огромными). Поэтому мы часто хотим [«уменьшать масштаб»](https://developer.android.com/topic/performance/graphics/index.html) изображений по мере их загрузки в память. Кроме того, обработка изображений может занять некоторое время, поэтому мы хотели бы делать это вне основного потока (например, в AsyncTask). Это может быть сложно; рекомендуемое решение — использовать стороннюю библиотеку, такую ​​как [Glide](https://github.com/bumptech/glide) , [Picasso](http://square.github.io/picasso/) или [Fresco](http://frescolib.org/) .

**10.6 Обмен файлами**

Как только у нас появится файл, хранящий изображение, мы сможем поделиться этим изображением с другими приложениями!

Как всегда, для взаимодействия с другими приложениями мы используем Intent. Мы можем создать *неявное намерение* для ACTION\_SEND, отправив сообщение всем приложениям, которые могут отправлять (делиться) изображениями. Мы установим тип данных как , image/\*чтобы отметить это как изображение. Мы также прикрепим файл как дополнение (в частности EXTRA\_STREAM). Еще раз обратите внимание, что мы на самом деле не помещаем *файл* в дополнение, а скорее **URI** для файла — тот, который предоставлен FileProvider!

*//java*

Intent intent = **new** Intent(Intent.ACTION\_SEND);

intent.setType("image/\*");

intent.putExtra(Intent.EXTRA\_STREAM, **this**.pictureFileUri);

**if** (intent.resolveActivity(getPackageManager()) != **null**) {

startActivity(intent);

}

Если вы делитесь файлом, который обычно является приватным (например, из Внутреннего хранилища или частного Внешнего хранилища), вам также нужно убедиться, что у «целевого» Activity есть разрешение на чтение файла. Вы можете предоставить это, добавив «флаг» к намерению:

intent.addFlags(Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION);

И благодаря этому вы сможете сделать снимок, сохранить его публично (или конфиденциально), но при этом поделиться им со всем миром!

1. <https://developer.android.com/training/basics/data-storage/files.html#InternalVsExternalStorage>[↩](https://info448.github.io/files-and-permissions.html#fnref34)
2. <https://developer.android.com/reference/java/io/File.html>[↩](https://info448.github.io/files-and-permissions.html#fnref35)
3. <https://developer.android.com/guide/topics/permissions/normal-permissions.html>[↩](https://info448.github.io/files-and-permissions.html#fnref36)